ARMY RESEARCH LABORATORY

Asymmetric Core Computing for U.S. Army High-Performance Computing Applications

by Dale Shires, Song Jun Park, Brian Henz, Jerry Clarke, Lam Nguyen, and Kelly Kirk

ARL-TR-4788 **April 2009**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-4788 **April 2009**

Asymmetric Core Computing for U.S. Army High-Performance Computing Applications

Dale Shires, Song Jun Park, Brian Henz, Jerry Clarke, Lam Nguyen, and Kelly Kirk
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
April 2009	Final	October 2007–September 2008

4. TITLE AND SUBTITLE
Asymmetric Core Computing for U.S. Army High-Performance Computing Applications

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dale Shires, Song Jun Park, Brian Henz, Jerry Clarke, Lam Nguyen, and Kelly Kirk

5d. PROJECT NUMBER
9UE11C

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRD-ARL-CI-HC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-4788

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
High-performance computing (HPC) is in a state of transition. HPC users have traditionally relied upon two things to supply them with processing power: speed of the central processing units (CPUs) and the scalability of the system. There are problems with this approach. Physical limitations are curtailing clock speed increases in general-purpose CPUs, the von Neumann load-execute-store approach does not map well to every computational problem, and systems of thousands of processors might be very inefficient depending upon processor interconnection limitations. Several versatile, commodity-based options are coming on line that could help address these deficiencies. Options now include throughput architectures such as graphics processing units (GPUs), reconfigurable systems built on field programmable gate arrays (FPGAs), multi- and many-core x86-based systems, and heterogeneous systems such as the Cell processor (incorporating a standard CPU and vector processing units). Each of these can be used to provide performance that, at one time, was only available by using Application Specific Integrated Circuits (ASICs) or large-scale fixed HPC assets. Newer methodologies hold out the hope of being more cost efficient and deployable along with providing faster deployment and development times and allowing the use of algorithms that remain modifiable at all stages of development and fielding. This report discusses our research on blending these asymmetric computing resources and addresses their use from the U.S. Army HPC perspective. We focus on the different methodologies and discuss performance from the perspective of kernels and applications that are relevant to fielding HPC technologies that will benefit the U.S. Army warfighter.

15. SUBJECT TERMS
graphics processing unit, GPU, asymmetric core computing, reconfigurable computing, HPC, heterogeneous computing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dale Shires
Unclassified	Unclassified	Unclassified	UU	32	19b. TELEPHONE NUMBER (*Include area code*) 410-278-5006

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

The authors wish to thank Dr. David Richie of Brown Deer Technology and the User Productivity Enhancement and Technology Transfer effort of the DOD High Performance Computing Modernization Program Office (HPCMO). Dr. Richie is an on-site researcher located at the U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, in the area of Electronics, Networking, and Systems/C4I. He provided valuable insights into new Application Program Interfaces, optimization approaches, and directions for graphics processing units (GPUs) being produced by AMD.

The authors also wish to thank Dr. Patrick Hanrahan and his students at Stanford University. Dr. Hanrahan is the principal investigator for the Technical Area 4 – Enabling Technologies component of the U.S. Army High Performance Computing Research Center. Dr. Hanrahan and his team provided valuable insights into the Cell processor and the inner workings of GPUs.

We are grateful to Mr. Daniel Pressel for providing initial implementations of the median stack calculation benchmarks.

The authors also wish to thank the Department of Defense HPCMO and the U.S. Naval Research Laboratory for time and assistance with using the Cray XD1 supercomputer.

1. Introduction

The high-performance computing (HPC) industry has relied on clustered computing (i.e., parallelism) of commodity processors and performance gains in central processing units (CPUs) that have tracked Moore's empirical law to field faster and faster systems. However, several issues are becoming pronounced that are limiting the effectiveness of this approach. First, scalar processors have for the most part followed the von Neumann architecture where a CPU follows a fetch, execute, and store instruction path. Mapping computationally complex algorithms to this sequence can have a limited effectiveness (many algorithms achieving only 10%–30% of peak theoretical efficiency). Second, the "free ride" on computer performance gains in the high performance computing world appears to be nearing an end. Dynamic power for a complementary metal oxide semiconductor (CMOS) chip is proportional to the product of load capacitance, the square of voltage, and switching clock frequency. Therefore, higher clock speeds of CPUs lead to more and more generation of heat. To compensate, chip manufacturers began shifting to more cores clocked at lower speeds (figure 1). This technique, known as voltage scaling, provides a throughput increase by using parallelism with no increases in power requirements (*1*). Parallelism is now the norm for developers who can no longer count on increasing capabilities and speeds of single processors (*2*). These changes have led to a large number of solution options for those seeking high performance.

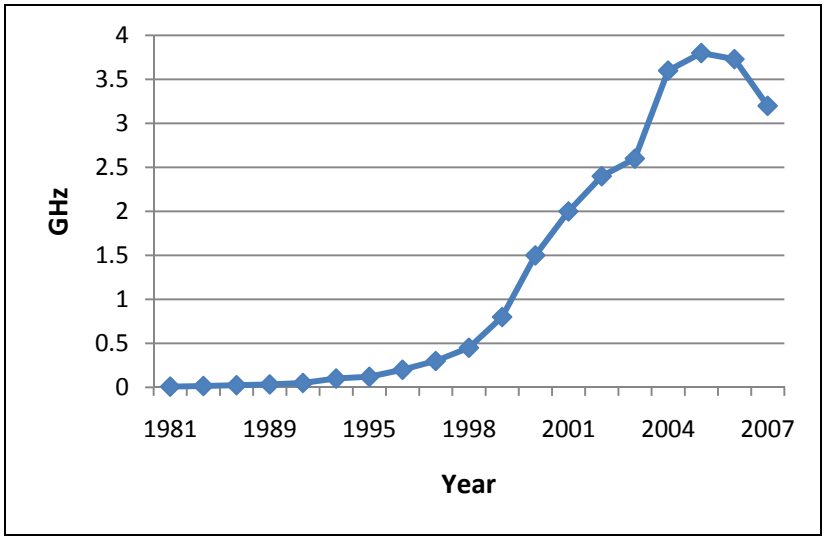


Figure 1. Clock frequencies of Intel processors.

Recognizing the shifting HPC landscape, we have engaged in a research program to assess how advances in the computing field can be applied to U.S. Army-relevant problems. While these technologies can be applied to a wide-range of physics-based codes, the primary focus is on battlespace applications to move HPC closer to Soldiers and field commanders. Technologies associated with information sciences, network sciences, and sensor systems to support the "every Soldier as a sensor" concept are the major driving forces for U.S. Army Transformation that includes Future Combat Systems (FCS). Computational sciences research along with computing power is essential to realize this transformation. Exponential growth in computing hardware and associated computing power over the last three decades provided the enabling technology to address a wide class of U.S. Army critical applications; however, we have not been able to harness the computing power to support operations on the battlefield due to the large foot print requirements of traditional high-performance computing systems. Current and future forces will rely upon network and battlefield information assimilation in complex urban environments to enable the Soldier to accomplish the mission. For example, processing information for situational awareness or understanding dynamic behavior of mobile ad hoc networks and associated sensor information on the battlefield involves handling terabytes of data. Integrating this data into simulations for actionable decisions to support field commanders will require moving HPC technology onto the battlefield. That is, we need substantial computing capability and associated software on the battlefield to support FCS and similar operations with confidence.

This research is assessing how the changing landscape in computing can be harnessed for maximum deployable U.S. Army benefit. Options we are investigating include multi-core processors, field programmable gate arrays (FPGAs), general-purpose graphics processing units (GPUs), and heterogeneous cores such as the Cell processor. These options allow for smaller footprint systems with tremendous speedups compared to traditional large clusters of von Neumann general-purpose CPUs. Our goal is to combine the asymmetric capabilities of these various cores into a complete high performance computing system. The primary focus is on battlespace applications to move HPC closer to the Soldier and field commander. Doing so opens up new possibilities to increase the capabilities of the Soldier and the systems being used in many application areas (e.g., sensors and intelligence applications). Algorithm development for applications needing improvements in speed and/or fidelity will be critical.

2. Relevant Technologies

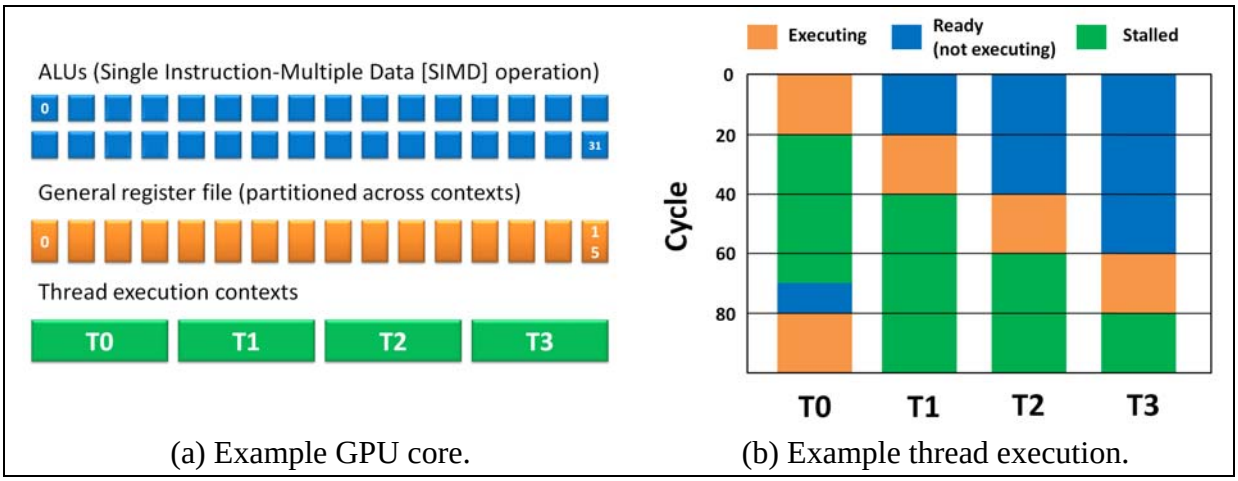
With the trend of decreasing price and size of processors, it is easy to envision today's supercomputer capability on a workstation in about a decade's time. On the other hand, associated software to take advantage of computing power and the communication bandwidth to

transfer gigabytes of data in the dynamic environment of the battlefield are not following this trend. Hence, PetaFlop computing is feasible within the next decade but software may not catch up to take full advantage of the hardware, especially in battlefield applications. The current best option to achieve a fielded and mobile HPC system is through the use of throughput architectures that combine disparate, asymmetric cores into a unified solution. These systems are focused on raw floating-point performance (with some attention to integer-based applications as well), and the application program interfaces (APIs) associated with them are rapidly maturing.

Throughput architectures come in various configurations and instantiations. Just about all of them have their origins in the commercial sector where virtual world simulation found in gaming engines requires large FLOP rates. At the bottom layer of execution, throughput architectures rely on single instruction multiple data (SIMD) parallelism. The SIMD execution model exploits data-level parallelism by operating a single instruction on different data sets. By having multiple SIMD capable processors on a chip, throughput architectures can execute independent SIMD instructions in parallel.

Hardware multi-threading attempts to hide the effect of stalls by letting the processors do some other productive task. This technique allows the general-purpose use of GPUs to be promising for applications with large amount of threads. GPU processing cores are heavily multi-threaded; typically requiring thousands of threads per core to achieve good performance. Two main factors can cause slowdown in these systems. The first deals with branches and data dependencies in code. Codes with many switch statements and conditionals, as well as temporal dependencies, can lead to under utilization of available computing units while waiting for results to be computed. The second is far worse and deals with code that has to access "remote" memory. These stalls can cost hundreds to thousands of cycles. Creating a large number of threads helps to cover these stall delays by allowing the compute core to switch to a new thread when one stalls. Different memories are available at different speeds, thus it is usually up to the application developer to pay close attention to memory access patterns to achieve good speedup. Compilers optimizations to do this task are slowly making progress.

Figure 2 illustrates a simple GPU core and how threads are executed (context switching) to keep the core busy (*3*). The core (figure 2a) can execute an instruction from one thread for each clock cycle but can maintain thread states for four threads. With the ALUs acting as SIMD processors, each can execute a vector-type instruction in a single clock (in this case, 32 concurrent instructions). Floating-point general-purpose vector registers are available and partitioned among the thread contexts. At runtime, the GPU runs a copy of the executable on each of the four thread contexts (figure 2b). T0 is executed until a stall is detected at cycle 20 (resulting from a memory reference that needs to be fetched or an instruction to complete). A context switch occurs to allow T1 to continue at cycle 20. T0 becomes ready again at cycle 70 and begins to execute again at cycle 80 after the stall by T3. Notice that this schedule allows all 32



(a) Example GPU core. (b) Example thread execution.

Figure 2. Simplified GPU and thread execution.

ALUs to be busy during the entire run. Also notice conceptually the large number of threads that are required to be running to allow the runtime system to efficiently keep the ALUs busy through context switching.

The Cell processor is an example of a chip containing heterogeneous multi-cores. The Cell has one main processor called the power processing element (PPE) and eight co-processors named synergistic processing elements (SPEs). Each SPE is a self-contained and independent processor capable of SIMD computations. Put another way, each SPE can run a distinct program. A notable difference in the Cell architecture is the absence of cache in SPEs. Instead, 256 KB of local storage space is available for each SPE and data must be transferred via data memory access (DMA). A common problem with these types of architectures deals with codes experiencing a lack of arithmetic intensity. Applications that do not have heavy floating-point calculation needs will not perform well on these systems. Another related issue is the amount of on-chip memory available to each core. If data is used faster than it can be loaded, the processors will stall while waiting for more data transfers to complete.

Another technology option at our disposal is reconfigurable computing. While scalar processors typically follow the von Neumann architecture approach of fetch, execute, and store, reconfigurable computing refers to processing with the aid of programmable logic, usually in the form of an FPGA. With an FPGA, data path and control flow can be modified in hardware as necessary to reduce an algorithm's execution time. Instead of computing through a series of instructions, a series of logic gates is created to solve a problem, thus closely matching the algorithm to the underlying hardware (figure 3). By coupling an FPGA with a processor, compute-intensive applications can be off-loaded from a host CPU to an optimized architecture in an FPGA. This integration allows an FPGA to function as a powerful co-processor and collaborate with a main CPU for performance acceleration. FPGAs are not really programmed

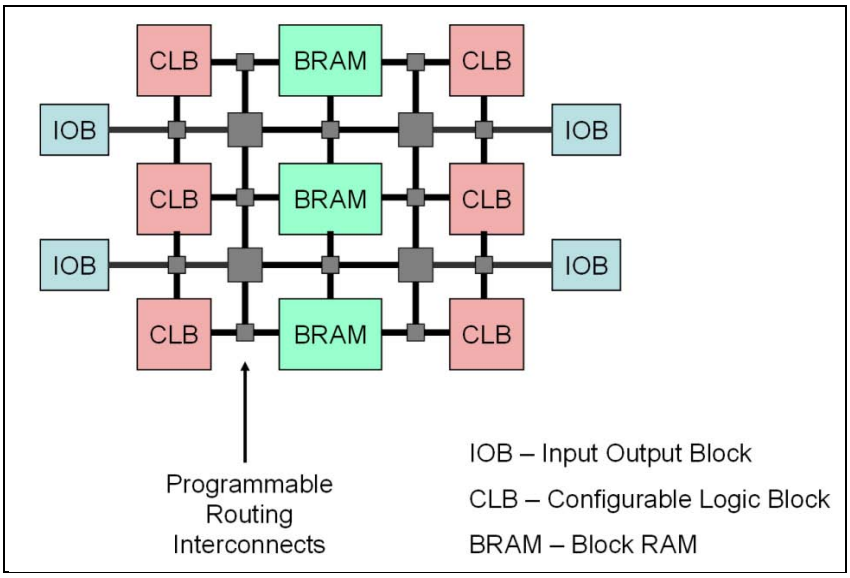


Figure 3. FPGAs provide a flexible fabric for circuit design.

in the traditional sense of the word. Rather, they require a hardware description language that has the concepts of concurrence built in. Due to the time-intensive development workflow required from hardware descriptions, several vendors are developing and refining procedural programming languages that down-compile to hardware implementations or descriptors like Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL).

Since most FPGAs are built on Static Random Access Memory (SRAM) technology, they have much slower clock rates than CPUs. Accordingly, they can only compete with CPUs when enough fabric is available to create multiple processing elements that can be pipelined. For example, assume a CPU clocked at 4 GHz compared against an FPGA at 400 MHz. Also assume that the CPU takes one cycle to produce some result. Given this scenario, the FPGA logic would have to be structured to allow for at least 10 “processing elements” to be created in order to equal the performance of the CPU.

3. Technical Approach

Our approach involved identifying relevant technology and establishing a laboratory to field, test, and investigate the various emerging components. These technologies are now part of an Asymmetric Core Computing Laboratory that has been established. The fielded components in the laboratory continue to evolve as the various technologies mature. Dual- and quad-core Intel-based processors are currently being used. To better understand how reconfigurable computing hardware can be applied, several FPGAs have been installed as well, including the Nallatech

H101-PCIXM board that provides a general-purpose framework for development and testing of reconfigurable computing. Two different GPUs have been installed. These include the NVIDIA GeForce 8800 GTX with 128 cores and AMD's FireStream using 320 cores. Two Sony Playstation 3 systems that feature the heterogeneous Cell as the main compute engine were procured and configured to run a version of Linux. The Cell has eight vector processing units controlled by a moderately powerful PowerPC chip. Some of these components were combined into a flexible architecture in the lab that contains a dual-core processor, FPGA board, and a GeForce GPU to provide a combined theoretical performance of 562 GFLOPS with the three distinct asymmetric core resources.

With the relevant technology identified, we moved toward addressing several computational kernels that would (a) help us learn more about the technology and (b) be relevant in their own regards to applications considered for future implementation on these resources. These small- to moderate-sized kernels provided a convenient way to understand the various programming languages and APIs used by the technologies. We also began working on a fielded system that could directly benefit from the technologies we were investigating. Since asymmetric cores require a careful partitioning to map applications to the optimal cores, we also spent a fair amount of time profiling codes and studying the codes' "signatures" to map the various pieces of the algorithm to the best architecture. Determining this partitioning *a priori* is not yet automatic and can be time-intensive.

As with all new technologies, several technical barriers must be addressed as these systems are applied to Army problems. Some of these are long-term and will be addressed over time as the technology matures. Others were more immediate as the technology was fielded to our targeted applications in the near term. Common across all of the technologies is the rather immature, although rapidly evolving, software development environments for the hardware solutions. AMD's SDK provides access to the intermediate Compute Abstraction Layer (CAL) low-level code generated by its compiler to allow motivated developers the option to try to do better than the compiler's back end code generator. Anecdotally this highlights that more work is often required for the compilers to be considered optimizing or robust. Furthermore, many of the languages being offered to developers remain proprietary acting as a "black box" filter with little or no details on how performance is being achieved. With FPGAs, there are noted performance issues with high-level language approaches.

Hardware issues are also evident. While the memory densities in FPGAs have increased, the use of floating-point units takes up a large amount of fabric space (worse for double-precision) that limits the number of processing elements that can be created. This in turn limits the size of the pipeline (or the number of processing elements) that can form and reduces performance. Single- vs. double-precision on GPUs remains an issue that developers must deal with. Early generation boards only supported single precision. Double precision is now also possible but this does impact the overall runtime. For example, on the AMD FireStream 9270, single precision

floating-point arithmetic performance is 1.2 TFLOPS. In contrast, double-precision speeds drop by a factor of 5 to 240 GFLOPS (*4*). Verification and validation are software engineering practices that developers must remain aware of as these technologies stabilize.

4. Research and Development Highlights

4.1 Cell Processor

One of our first attempts at using a heterogeneous system involved the Cell processor. The Cell was of interest due to the coupled system of a general-purpose CPU (PowerPC) and vector processing units known as Synergistic Processing Elements (SPEs). There was a record of researchers achieving speedup using the processor, although the results and level of performance were somewhat varied (*5*, *6*). We first worked with Open Computer Vision (OpenCV) on some real-time edge detection algorithms. We also investigated the Sequoia approach from Stanford and the Cell Messaging Layer; both available APIs to program the Cell. Sequoia is a portable, low-level language that attempts to map to superscalar and parallel machines through C-like extensions (*7*).

However, the Cell has several issues that caused us to not pursue it more vigorously, at least in the near term. There has been some distinction made between multi-threaded architectures and streaming architectures, the later focusing more on the explicit attention a user must pay to establish communication streams vice computation streams (*8*). The Cell can be thought of as an example of a streaming processor; a DMA is used to move data from the main memory to the local storage space. A stall in any SPE is extremely costly. Effective streaming poses many difficulties, including finding applications that can effectively work with the no cache architecture and limited memory. Furthermore, we engaged in several discussions with our research partner Stanford University (through the U.S. Army High Performance Computing Research Center [AHPCRC]) who had experimented with the Cell in several projects. We learned several things from this discussion including the incredible level of effort they had to expend to achieve speedups using the system and the difficulty of using the Cell API. Furthermore, they provided more details on the difficulty of overcoming the limited memory available to the SPEs. While the SPEs are efficient by having no cache and a fast access scratchpad memory, it makes programming very complicated (*9*). Other issues, notably main memory access rates and a rudimentary programming paradigm have been noted when trying to use the system for HPC cluster computing (*10*). This will certainly be revisited as necessary, largely based on what decisions will be made for a processor to populate a Playstation 4 (should one be announced).

4.2 FPGAs

Reconfigurable computing refers to performing computations using Field Programmable Gate Arrays (FPGAs). An FPGA is a chip that flip-flops, allowing designers to program its reconfigurable architecture to suit a specific problem at hand. FPGAs attempt to combine the advantages of fast Application-Specific Integrated Circuits (ASICs) and general-purpose CPUs. CPUs compute via a series of instructions, whereas FPGAs compute via a series of connected logic gates. FPGAs can be conceptualized as reconfigurable ASICs. Reconfigurable computing is taking a hardware approach to performing calculations. Traditionally, FPGA computing was beneficial in special applications involving high degrees of bit manipulations (encryption) or applications requiring field programmability (e.g., space applications). CPUs dominate the field of computing, but recent trends indicate a shift toward multi-core, parallel architectures. This implies that CPUs can promise increased parallelism, but not increased single-thread performance. Because an increase in performance is not automatic with newer hardware, software development becomes critically important. Without sufficient software to leverage its full potential of the parallel hardware, even the most powerful hardware becomes inconsequential. This research assessed programming aspects of FPGA development to allow us to determine realistic performance of FPGA co-processors for integer- and floating-point based applications.

Hardware description languages, mainly VHDL and Verilog, are used for developing tailored hardware designs for FPGA implementation. This method requires background knowledge in digital circuits involving electric signals and logic devices. In an effort to field their technologies into newer and more diverse areas, FPGA vendors are trying to overcome the barriers found in programming their devices by using high-level language development approaches. High-level languages raise the abstraction level, which in effect relieves developers from the FPGA's low-level details. Unfortunately, due to a lack of standards, these languages are vendor-specific and hardware-dependent. Due to time constraints and availability, we focused on only two vendor approaches (DIME-C and Mitrion-C). The main trade-offs related to high-level languages are abstraction and access. Conceptually, a higher abstraction level makes programming both easier and faster. However, visibility or access to underlying hardware, that has potential for optimization, is lost. Our overall impressions with the three approaches are shown in table 1.

Table 1. FPGA development time estimates.

Development Stage	VHDL	DIME-C	Mitrion-C
Background learning	High	Low	Medium
Writing source code	High	Low	Medium
Debug and simulation	Low	High	Medium
Applying design changes	High	Low	Medium
Maintaining	High	Low	Medium

Two hardware systems equipped with an FPGA co-processor were utilized in this research. First, local Linux machines were outfitted with Nallatech H101PCIXM boards featuring Xilinx Virtex-4 LX100 connected to the host via a PCI-X bus. Second, a Cray XD1, a system provided by the DOD High-Performance Computing Modernization Program (HPCMP) residing at the Naval Research Laboratory, offers Virtex-II ProVP50 and Virtex-4 LX160 additions to the system.

The algorithms studied included the integer-based Blowfish encryption hashing function and floating-point median calculation with a bubble sort at its core. The hashing technique follows the OpenBSD password authentication where the hardware design is set to perform a brute-force attack. Median calculation, derived from real-world signal processing of noisy data, evaluates the feasibility of floating-point operations on modern FPGA systems. Median values are determined by applying the bubble sort routine (used for simplicity) to a list of numbers.

The encryption algorithm was implemented on various platforms and languages. Performance results are measured in terms of throughput (number of keys tested per second). Details on the hardware used and the results achieved are available in table 2 and figure 4. A late addition to this study was the use of a multi-threading GPU. The NVIDIA GeForce 8800 GTX was used for this with the software being written using the Compute Unified Device Architecture (CUDA) language. The ability to incorporate low-level hardware optimizations using VHDL allowed us to achieve best performance. While they do allow for quicker fielded solutions, higher-level approaches incorporate an overhead associated with automatic translation that proved to be damaging to performance.

Table 2. Blowfish hardware details and performance comparison.

Language	**Processing Hardware**	**Clock Frequency (MHz)**	**One Unit Execution Time (µs)**	**Processing Units**	**Throughput (keys/s)**
ANSI C	Xeon	3000	54	1	18,518
VHDL	Virtex-4 LX100 FPGA	65	120	11	91,666
DIME-C	Virtex-4 LX100 FPGA	51	1850	4	2162
Mitrion-C	Virtex-2 Pro VP50 FPGA	100	822	2	2433
	Virtex-4 LX160 FPGA	100	823	4	4860

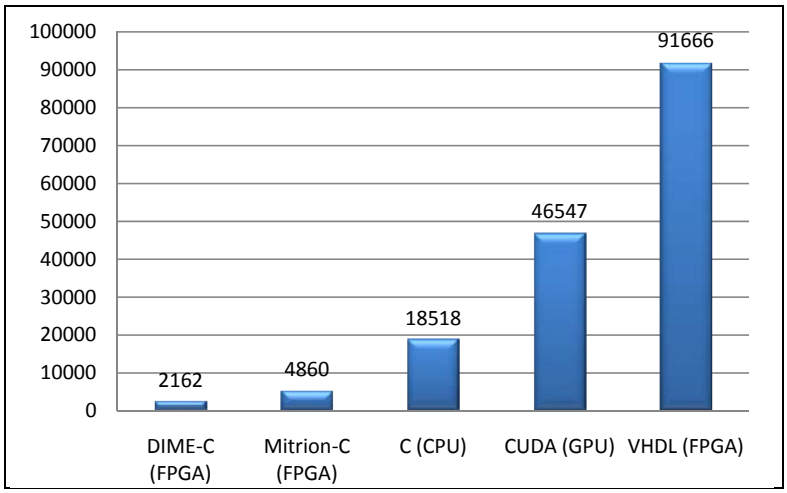


Figure 4. Blowfish throughput performance (keys/second).

The very simple FPGA-based median value floating-point algorithm failed to outperform a general-purpose Intel processor (figure 5). This is primarily because floating-point units require more resources. For example, a 32-bit integer adder requires 32 slices whereas a 32-bit floating-point adder requires approximately 380. This increased hardware fabric requirement severely limits the number of processing elements that can be formed in the FPGA. Since FPGAs are built on SRAM technology, they have much slower clock rates than CPUs. Accordingly, they can only compete with CPUs when enough fabric is available to create multiple, pipelined processing elements or when the algorithm executes poorly on a CPU general-purpose architecture. Currently, floating-point area requirements are too expensive for efficient FPGA implementations. Due to time constraints, a more detailed and customized solution using VHDL was not attempted.

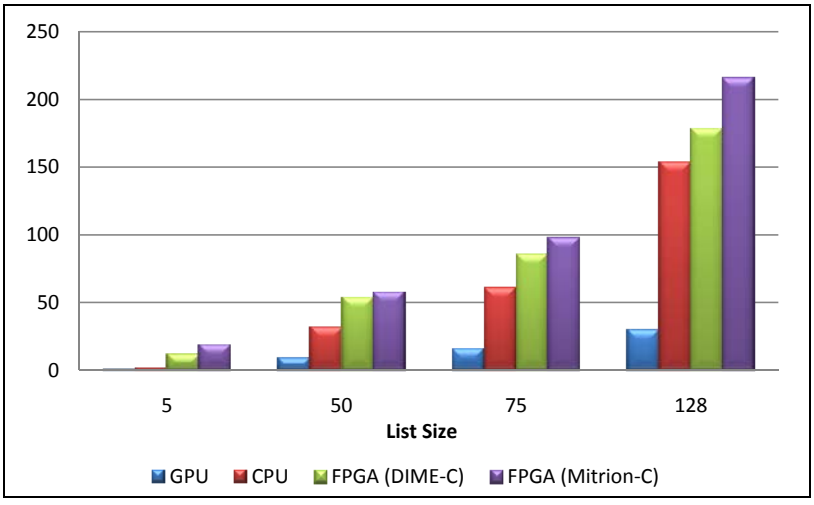


Figure 5. Median value (floating-point) calculation requirements (seconds) using various list sizes.

FPGAs provide versatility, custom dedicated design, and low-power operation. Adaptable hardware performance can be achieved for applications demanding high computational requirements. Compared to ASICs, the reconfigurable characteristic of an FPGA reduces time to market since the fabrication process is not necessary. As for potential impact to the U.S. Army, field-dependent and compute-intensive applications with rapidly changing requirements can benefit from utilizing FPGA technology. One favorable area is information processing with its computationally challenging problem size due to large amounts of data acquired by ever increasing numbers of sensors. Possible applicable areas would be in signal processing for combat vehicle systems (e.g., radar processing systems). Driver-assistance applications such as adaptive cruise control, target recognition, collision warning, and lane warning would require higher processing capabilities requiring parallel signal processing. Here, the flexibility of FPGA systems can support evolving algorithms resulting from changes in mission and terrain environment.

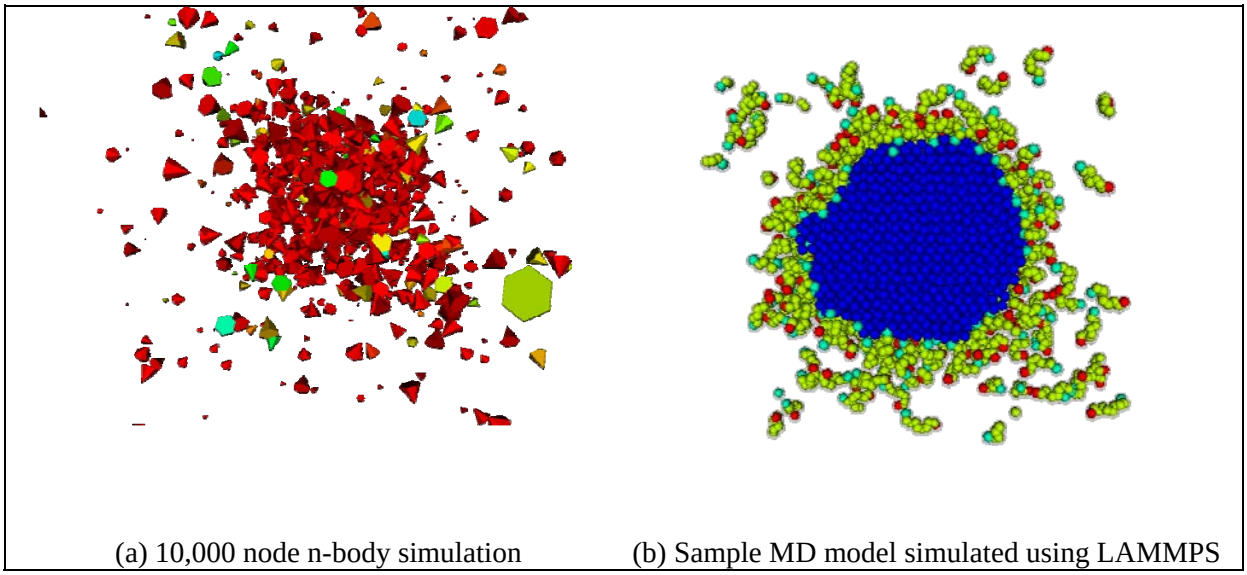
4.3 GPU

4.3.1 N-body

Several computational kernels were implemented on GPUs. The first was the n-body method used to solve Newton's laws of motion for large numbers of interacting bodies. These methods are used in many applications of interest to the DOD and U.S. Army including molecular dynamics (MD), particle physics, and plasma physics. The computational requirements for these methods increase as the square of the number of interacting particles, i.e., $O(n^2)$ so that a system with 20 points takes 4× as many computations as a system with 10 points. Traditionally these algorithms have used parallel processing techniques to simulate ever larger model systems, and they scale quite well to thousands of processors. In many of these methods, the computational kernel is small but heavily utilized. This results in a method that is ideally suited to the use of GPUs for acceleration.

In this effort, two applications have been considered, namely, an astrophysical simulation and an open source MD code. The first of these, the astrophysics simulation, is completely re-implemented within a GPU framework for maximum acceleration. The second application, Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) from Sandia National Laboratories, is accelerated by offloading the intensive computational kernel to the GPU while maintaining the original code structure. Both of these approaches have advantages and disadvantages that will be discussed. Sample output from both codes is shown in figure 6.

Since there are many n-body algorithms consuming large numbers of cycles on DOD supercomputers, one immediate impact of accelerating these applications is that many hours of CPU time could be offloaded to GPU systems such as scientific visualization clusters that would normally house high-end GPUs. This would free up the more general purpose CPUs on current cluster systems for other applications that are not as easily ported to GPUs. For instance, the



(a) 10,000 node n-body simulation (b) Sample MD model simulated using LAMMPS

Figure 6. N-body simulation results on a GPU.

port of the LAMMPS application to GPUs resulted in an overall speedup of about 4.5, so that eight GPUs could do the work of 36 CPU cores. The astrophysics application showed even greater speedup of about 60 so a small system with 4 GPUs could match the performance of 240 CPU cores. A comparison of the CPU and GPU is shown in table 3 for different numbers of simulated bodies. In the future, this will allow for the utilization of these algorithms in a small form factor, portable architecture that could be used on or near the battlefield.

Table 3. Astrophysics n-body simulation comparison.

No. of Bodies	**CPU Kernel (s)**	**GPU Kernel (s)**
1,000	0.2	0.01
5,000	4.0	0.09
10,000	18.8	0.32

N-Body algorithms have many characteristics that make them potential candidates for acceleration using GPUs. These include large numbers of independent computations, i.e., each atom interacts with many other atoms, high floating point operation (FLOP) counts, and fine granularity. This last item is due to the fact that the interactions can be computed independently rather than being cumulative. In n-body simulations, the majority of time is typically spent computing the potential function, or computing particle interactions. In computing the potential function the distance from all neighboring atoms must be computed, a task that is $O(n^2)$. A commonly used potential in the MD method is the Lennard-Jones potential as described by equation 1.

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], \quad (1)$$

where *V(r)* is the potential energy, ε is the potential well depth, σ is the distance at which the potential is 0, and *r* is the distance between atoms. For improved performance, a cutoff distance is used, outside of which the particle interactions are effectively 0. An efficient force computation therefore requires a neighbor list that stores the particles within the cutoff distance of each other.

One interesting implementation tradeoff for LAMMPS on the GPU is that although the interactions for atom pairs is symmetric, so that $dV(r_{i,j})/dr = -dV(r_{j,i})/dr$, it is faster to recalculate the force acting on $r_{j,i}$. This is because the access to main memory in order to copy the results requires many more cycles to complete than the actual computation. Taking this into consideration, we note that the GPU is performing about 2× the number of floating point operations that the CPU does and is still achieving a 4.5× overall speedup. Another performance issue is the transfer rate of data from main memory to the GPU. This occurs over the PCI-Express bus with the GeForce 8800 GTX video card having a transfer rate of about 86 GB/s. This factor must be taken into consideration when large datasets must be transferred to the GPU for computation. If the amount of computation does not offset the data transfer time, it will not be efficient to use GPUs.

By using GPUs it has been possible to improve the performance of an optimized molecular dynamics application by a factor of 4.5 all the way to a factor of 60. As we can see from the acceleration results, the improved performance of an application can approach the theoretical speed-up based on peak FLOP rates if one is inclined to completely rework an application into the GPU framework. This is labor intensive and primarily feasible for relatively small code bases, unless a significant investment is warranted. On the other hand, accelerating a specific portion of an application is relatively easy to do so long as the computational load far outweighs the communication requirements (and the computations are floating-point based). GPUs present researchers with a unique piece of hardware for developing floating-point intensive applications, and GPUs are fairly low cost components because of the massive gaming industry that requires ever better graphics and performance, driving down the cost of off-the-shelf hardware.

4.3.2 Monte Carlo

Another interesting kernel that appears in many codes requiring a global optimum solution is the Monte Carlo method. We chose to investigate this algorithm through a computational stereo matching process to produce a very fine, globally optimized disparity map where every pixel in one image is matched with its corresponding pixel in the stereo pair. This method is also quite tolerant of slight variations in the image pair and can deal well with occlusion (areas not capable of being perfected matched due to obstructions in the image pair). Just as with other Monte Carlo algorithms, this approach requires a significant number of floating-point operations.

However, the process of matching pixels typically requires only local interactions. On a digital computer memory, this translates into local memory references since the images can be mapped to contiguous memory blocks. Furthermore, the amount of processing for each pixel remains uniform. All of these factors lead to an approach that is promising for effective GPU implementation.

Computational stereo matching describes the process of synthesizing the mechanics of binocular vision. Stereo camera pairs are aligned along the *y* and *z* axes and slightly offset in the *x* axis (horizontal). Because of binocular parallax, these cameras (referred to as the left and right camera pair) will acquire slightly different images of the scene due to the horizontal shift. A point *P* in the three-dimensional (3-D) scene is projected onto the left and right camera photosensitive plates at P_L and P_R, respectively. The disparity is the difference in the locations of P_L and P_R that results from the camera shift in the *x* axis. This disparity, along with details of the camera optics system, can be used to determine *P*'s position in the world. For example, a point very distant from the cameras will appear to be at the same vertical and horizontal position on monitors attached to the left and right camera pairs. However, a point close to the camera pair will not be in the same position on the monitors. Rather, the point on the left monitor will be displaced (the disparity) to the right from the corresponding point on the right monitor.

Once the matching points are found in an image, an inverse perspective transform or simple triangulation can be used to derive the two lines where the projection of the world point strikes the photosensitive plate of the camera. When these lines are intersected, the 3-D characteristics of the scene can be recovered. The ability to properly match stereo camera pair images is important to any application in which distance or range information must be extracted from an image. One such application is plotting terrain contours from images shot by aerial camera pairs. Computer vision, biometrics, and robotics are also fields where this technique is important. The system is also passive; it does not require active sensing devices that could be problematic for the mission requirements. This technique also provides a way to perform image registration. Image registration attempts to match images taken at different times and can provide details on changes that might have occurred.

This process requires a global optimization. Since the digital image data maps pixel intensities to a relatively low resolution, there are many possible matches in the local sense. That is, swatches of one image may appear to map other portions of the stereo pair. To perform stereo ranging, the whole image must be taken into account. Simulated annealing provides an optimization technique to locate a global optimum. Annealing involves heating a solid and letting the molecules rearrange themselves. The temperature is then slowly lowered allowing the molecules to settle into a low energy state (thermal equilibrium). This algorithm mimics this process using a Monte Carlo search. A disparity map is maintained for the image pair and is randomized at the start in the range [0..maximum disparity]. Random, local state transitions in the map are performed by varying the disparity value only slightly. If the change in energy

(defined by a brightness and smoothness constraint) results in a lower energy level, it is accepted. If the delta energy is higher, the change is only accepted with a probability within the Boltzman distribution. The brightness constraint enforces the rule that matched pixels should have similar intensity (e.g., gray scale or color) values. The smoothness constraint asserts that the horizontal shift of an element should be roughly equal to that of its neighbors. This is necessary since the first constraint is strictly local and stereo correspondences are local ambiguous. Without this constraint, surfaces would not be spatially coherent (*10*).

An algorithm to perform the simulated annealing routine was developed and implemented on the GeForce 8800 GTX GPU using CUDA. Several test cases were validated on the GPU, including computer-generated random dot stereograms and an actual stereo pair as shown in figure 7. The results were impressive. Even with interactive visualization of the disparity map as it was forming, causing extensive PCI-Express bus traffic, the 8800 GTX code ran ~25× faster than a C version of the code running on an Intel Xeon 3.00 GHz processor. This algorithm also provided valuable lessons on memory management within the GPU and a better understanding of the maturity of the CUDA compiler.

Figure 7. A left and right stereo camera pair and resulting disparity map showing areas near camera (lighter) and areas distant (darker).

4.3.3 Obstacle Detection and Avoidance

To support the U.S. Army vision for increased mobility, survivability, and lethality, the Sensors and Electron Devices Directorate (SEDD) at the U.S. Army Research Laboratory (ARL) has designed and developed the forward-looking ultra-wideband (UWB) synthetic aperture radar (SAR) (test platform shown in figure 8). The radar is based on time-domain wideband impulses and uses a data acquisition technique called Synchronous Impulse REconstruction (SIRE) that

Figure 8. Prototype UWB SAR with synchronous impulse reconstruction (SIRE).

allows it to employ relatively slow analog to digital converters (ADCs) to digitize wideband signals. The configuration assumes that the radar and targets are stationary during the data acquisition cycle when, in reality, target signatures suffer distortions in phase and shape because of the radar motion. The phase error leads to significant loss in target radar cross-sectional values in resulting imagery and the shape errors destroy the frequency contents of the targets and thus the ability to discriminate targets from other objects. These errors are mitigated by a signal-processing method developed to recover the accuracy of the target signatures affected by radar motion. This approach was applied to simulated data and measured data from the SIRE radar. The result was perfect reconstruction using the simulated data and significant improvement in the resulting SAR image for the measured data (*11*).

The computational requirements for this processing are significant. Including data acquisition rates and integration using forward motion, along with parameters such as platform speed and sub-image size, the operation count quickly exceeds 20 million operations per second. These operations are distributed over several signal and image processing filters such as forward motion processing and backprojection image formation. The mathematics is predominately floating-point with additions, multiplications, divisions, and square roots. The high FLOP rate requirements made this an ideal candidate as we considered using asymmetric core computing techniques to increase computational performance for battlespace applications. Our goal was to use these technologies to speed the processing to achieve near or at real time speeds.

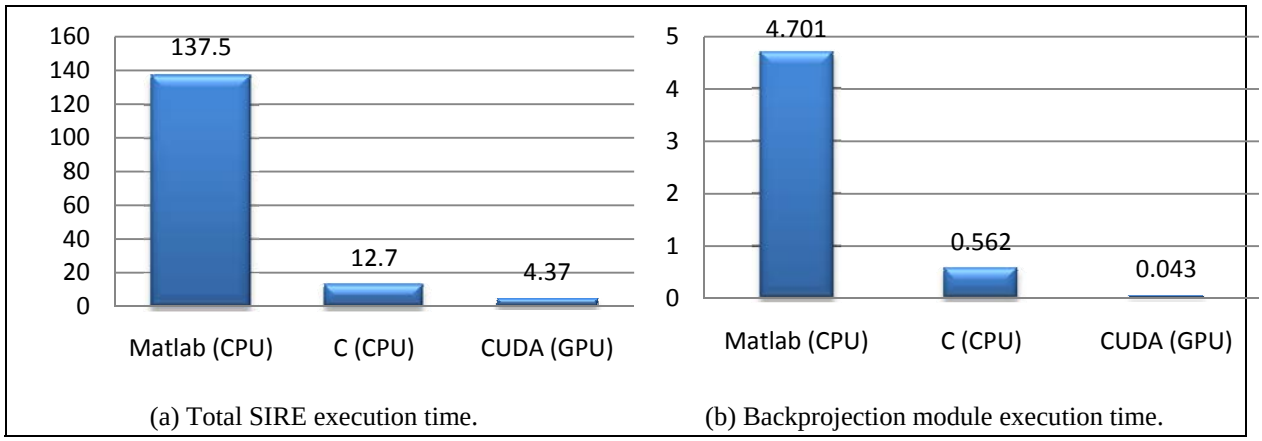
Due to the high floating-point requirements of this code, we decided to target the NVIDIA GTX 8800 GPU hardware and the associated CUDA API components in our Asymmetric Core Computing Laboratory for speedup. The 8800 GTX has 128 cores clocked at 1.35 GHz with a FLOP rate of about 345 GFLOPs. CUDA provides a “C”-like programming interface where kernels are written to run on the device (GPU) and are callable from the host (CPU).

Our starting point was an initial code written in Matlab. Matlab is widely used in the Signal and Image Processing (SIP) community and is best at providing a wide-range of functionality with an easy-to-use scripting language. However, execution speed is not one of its strong points. This code was deconstructed and re-implemented in the C computer language. This effort took a fair amount of work as several routines, such as the not-a-knot spline used in Matlab for curve fitting, had to be reconstructed in C. The code expansion factor (as determined from source lines of code) was about 3.75 since many Matlab calls required several lines of C code to produce the same functionality. A profile of the Matlab code and the C code showed roughly the same code characteristics. Over a small set of test frame data, about 60% of the total execution time in Matlab was spent performing the backprojection module (76% of total in C). The majority of the time is spent computing 3-D distance calculations. Given the fact that so much time is spent in the routine and the high rate of floating point instructions per branch taken, we chose to concentrate on this module for GPU implementation.

The 8800 GTX uses a fast context switching design for threads executing on the 128 compute cores. Accordingly, it is best to use a fine-grain approach to parallelism as one maps an algorithm to the device. Accordingly, one pixel of each 100×250 frame is assigned to a thread. If a thread stalls waiting for a computation to complete (such as a division operation requiring several clock cycles), the runtime system will context switch to a new thread so the core can continue to work. Device compilers are not yet mature, but during the execution of this project NVIDIA published a profiler to assist developers in finding clues to source code transformations that might help in reducing runtimes. Several CUDA source code optimization techniques were used to minimize branch divergence and maximum coalesced memory calls and on-chip memory use.

The CPU used (for Matlab and C) in these tests consisted of a dual-core Intel Xeon processor clocked at 3.0 GHz. The charts in figure 9 show the achieved performance for the overall SIRE code (a) and the backprojection module alone (b) of processing 32 m. Overall, the SIRE radar processing code using GPU acceleration is running at ~31× faster than the baseline code. The backprojection routine alone is computing at two orders of magnitude faster (~110×). The radar's data acquisition is performed with the platform moving at an average speed of about 1 m/s. The Matlab code was only performing at rates around 0.11 m/s or about 10× slower than real time processing. With the accelerated code, the current physical parameters are easily exceeded with a potential of about 3.5 m/s processing.

Further code speedup may be possible. The initial conversion was done in a sequential, control-flow approach where the C code follows a flow chart amiable to real time processing. This approach can limit GPU performance due to data transfers over the PCI-Express bus (frame data is moved from the CPU to the GPU for each call to the backprojection routine). It may be possible to restructure the C code if the system is to run in post-processing mode only. This could be done by loading more than one frame at a time on the GPU for parallelism across the frames and pixels rather than waiting for one frame to complete at a time.



(a) Total SIRE execution time. (b) Backprojection module execution time.

Figure 9. Wall clock times (in seconds) of the SIRE code using different languages and cores.

5. Emerging Directions

Software engineering research is vital to achieve the potential speedups provided by an asymmetric core computer. There are several concerns that will need to be addressed. New approaches will be required for algorithm design and analysis; traditional models might not always work. For example, on GPUs an *O(n)* work algorithm may actually require a longer run time than an *O(n log n)* approach. Redundant calculations are, in many cases, preferable to memory operations. The tradeoff in program development time and efficiency also needs to be better understood. Furthermore, historic optimization approaches as applied to superscalar von Neumann architectures, such as loop unrolling, may not always be beneficial. As long as some compiler, implementation, and runtime details remain hidden from developers, more work will have to be done to discover those transforms that might lead to an increase in performance. The field is currently wide-open and challenging. An integrated approach using computer science, computer engineering, and electrical engineering is required to deal with these new technologies in order to achieve an optimal solution.

Recently, many vendors are signing on to the Open Computing Language (OpenCL) effort (*12*). This standard is open and is designed to allow efficient use of heterogeneous computing resources. OpenCL has a software stack consisting of a platform layer, a runtime system for resource management, and a compiler. The platform layer allows a query and select system to allow algorithms to select the optimal computing device available (based on load and the application's signature). It also supports both the data-parallel and task-parallel paradigms for parallel computing. Hence, it is applicable to conceivably all parallel tasks.

Another interesting development in GPUs is the use of superscalar processing cores. This is another level of parallelism that has been added to the AMD FireStream cores. The raw floating-

point capacity of the 800 cores in the AMD FireStream 9270 is immense but not easily achieved. Maximum performance is often only achieved after careful code analysis and the targeted use of clever algorithms, tricks, and optimizations.

As we progress in this research, we will be looking at using more asymmetric cores (such as a combined FPGA, GPU, and CPU approach) in algorithms or parallelizing across numerous asymmetric cores. This will involve detailed algorithm analysis and selection of cores based on code signatures. The difficulties here are predominately load balancing, selection of either task or data parallelism, and a way to synchronize and perform data transfers. A simplified example of this is the Folding at Home project at Stanford where individuals can offer their idle processors (whether it is Windows PCs, Linux workstations, Playstation 3s, etc.) to a largely data parallel application. These issues will need to be addressed as various streaming and multi-threaded architectures find their way to fielded units such as the Company Operations and Intelligence Cells (COICs). To be fully effective, this will involve some research on work distribution and load balancing among the components.

Several questions will have to be addressed as we consider a runtime query system for load distribution and balance. How would resources characterize their capabilities and report them to the runtime system? How would task parallelism be represented as compared to data parallelism? How would network topologies connecting the various cores be represented? The general distributed data processing (DDP) problem will be studied as there are several tools and techniques that might be applicable. Different applications will probably be needed at the various layers. For example, data movement might be implemented by integrating numerous hosts using an approach such as the message-passing interface (MPI).

6. Conclusion

There is a fundamental change occurring in the HPC field as throughput architectures gain in importance, market share, and raw floating-point computational capacity. Portable and even hand-held HPC is quickly becoming a reality, and a concerted effort is needed to make these technologies viable to Army forces. We have been able to show a considerable amount of speedup in many applications and kernels. Unfortunately, the performance gains achieved are at times proportional to the amount of effort put into the task. New approaches are holding out some promise of moving these technologies more into the open source arena, thus allowing researchers more direct understanding and control at low levels. These efforts should greatly facilitate the maturation of compiler and API technologies. The performance of these new hardware approaches is providing a way to field HPC and we look forward to continued research and development efforts to transition these new technologies to Army applications.

7. References

1. Bhandarkar, D. *Multi-Core Microprocessor Chips; Motivation and Challenges*; Intel Corp., May 2006.
2. Koch, G. Discovering Multi-Core: Extending the Benefits of Moore's Law. *Technology at Intel Magazine* July **2005**.
3. Fatahalian, K.; Houston, M. A Closer Look at GPUs. *Communications of the ACM* **2008**, *51* (10).
4. AMD. *Ultimate Compute Performance*; FireStream 9270 Product Literature, 2008.
5. De Fabritiis, G. *Performance of the Cell Processor for Biomolecula Simulations*, Computational Biochemistry and Biophysics Lab: Barcelona, Spain, 2007.
6. Olivier, S.; Prins, J.; et al. Porting the GROMACS Molecular Dynamics Code to the Cell Processor. *IEEE Parallel and Distributed Processing Symposium*, 2007.
7. Fatahalian; Knight; et al. Sequoia: Programming the Memory Hierarchy. *Proceedings of Supercomputing*, Tampa, FL, November 2006.
8. Mattson; Lethin; et al. Stream Virtual Machine and Two-Level Compilation Model for Streaming Architectures and Languages. *Presentation at Language Runtimes '04*, October 2004.
9. Scarpazza, B.; Villa, O.; Petrini, F. *Programming the Cell Processor*; Dr. Dobb's Journal; 9 March 2007.
10. Barnard, S. A Stochastic Approach to Stereovision. In *Readings in Computer Vision*; Addison-Wesley: New York, NY, 1987.
11. Nguyen, L. *Signal Processing Technique to Remove Signature Distortion in ARL Synchronous Impulse Reconstruction (SIRE) Ultra-Wideband (UWB) Radar*; ARL-TR-4404, U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2008.
12. Munshi, A. OpenCL: Parallel Computing on the GPU and CPU; SIGGRAPH, 2008.

NO. OF COPIES	ORGANIZATION
1 (PDF only)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LAB IMNE ALC HRR 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI OK TL 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI OK PE 2800 POWDER MILL RD ADELPHI MD 20783-1197
	ABERDEEN PROVING GROUND
1	DIR USARL AMSRD ARL CI OK TP (BLDG 4600)

NO. OF COPIES	ORGANIZATION
1	PRGM DIR C HENRY 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
2	HPC CTRS PROJ MGR B COMMES CHF SCNTST D POST 10501 FURNACE RD STE 101 LORTON VA 22079
1	DIR USARL AMSRD ARL CI J GOWENS 2800 POWDER MILL RD ADELPHIA MD 20783-1197
1	J OSBURN CODE 5594 BLDG A49 RM 15 4555 OVERLOOK RD WASHINGTON DC 20375-5340
1	AIR FORCE RSRCH LAB MTRLS & MFG DIRCTRT R PACHTER AFRL MLPJ 3005 HOBSON WAY BLDG 651 RM 189 WRIGHT PATTERSON AFB OH 45433-7702
1	AIR FORCE RSRCH LAB K HILL AFRL SNS BLDG 254 2591 K ST WRIGHT PATTERSON AFB OH 45433-7602
1	AFRL IF R LINDERMAN 525 BROOKS RD ROME NY 13441-4505
1	NVL OCEANOGRAPHIC OFC OFC OF THE TECH DIR J HARDING CODE OTT STENNIS SPACE CENTER MS 39529

NO. OF COPIES	ORGANIZATION
1	INFO TECHLGY LAB US ARMY ENGR RSRCH & DEV CTR D RICHARDS VICKSBURG MS 39810
1	SPAWAR SYS CTR C PETERS BLDG 606 RM 318 53360 HULL ST SAN DIEGO CA 92152
1	ARNOLD ENGRG DEV CTR C VINING 1099 SCHRIEVER AVE STE E205 ARNOLD AIR FORCE BASE TN 37389
1	AIR FORCE RSRCH LAB SENSORS DIRCTRT T WILSON 2241 AVIONICS CIR WRIGHT PATTERSON AFB OH 45433
1	US ARMY RSRCH & DEV CTR NVL CMND CNTRL & OCEAN SURVEILLANCE CTR HPC COORDNTR & DIR DOD DISTRIBUTED CTR NCCOSC RDTE DIV D3603 L PARNELL 49590 LASSING RD SAN DIEGO CA 92152-6148
1	UNIV OF TENNESSEE ASSOC DIR INNOVATIVE COMPUTING LAB COMPUTER SCI DEPT S MOORE 1122 VOLUNTEER BLVD STE 203 KNOXVILLE TN 37996-3450
2	SOUTH CAROLINA STATE UNIV EXEC DIR LS SCAMP S ALLEY J GUYDON 300 COLLEGE ST NE PO BOX 7212 ORANGEBURG SC 28117

NO. OF
COPIES ORGANIZATION

ABERDEEN PROVING GROUND

19 DIR USARL
AMSRD ARL CI
R NAMBURU
AMSRD ARL CI H
C NIETUBICZ
B SHEROKE
AMSRD ARL CI HC
P CHUNG
J CLARKE
M LEE
D PRESSEL
D SHIRES
R VALISETTY
AMSRD ARL CI HM
P COLLINS
M KNOWLES
AMSRD ARL CI HS
L BRAINARD
D BROWN
R CAMPBELL
T KENDALL
K SMITH
AMSRD ARL WM
P PLOSTINS
AMSRD ARL WM BC
J SAHU
P WEINACHT

INTENTIONALLY LEFT BLANK.